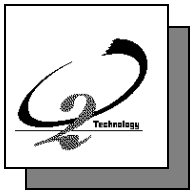


O_2Search
User Manual

Release 5.0 - April 1998

Information in this document is subject to change without notice and should not be construed as a commitment by O_2 Technology.

The software described in this document is delivered under a license or nondisclosure agreement.

The software can only be used or copied in accordance with the terms of the agreement. It is against the law to copy this software on magnetic tape, disk, or any other medium for any purpose other than the purchaser's own use.

Copyright 1992-1998 O_2 Technology.

All rights reserved. No part of this publication can be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopy without prior written permission of O_2 Technology.

O_2, O_2API, O_2C, O_2DBAccess, O_2Engine, O_2Graph, O_2Kit, O_2Look, O_2Store, O_2Tools, and O_2Web are registered trademarks of O_2 Technology.

SQL and AIX are registered trademarks of International Business Machines Corporation.

Sun, SunOS, and SOLARIS are registered trademarks of Sun Microsystems, Inc.

X Window System is a registered trademark of the Massachusetts Institute of Technology.

Unix is a registered trademark of Unix System Laboratories, Inc.

HPUX is a registered trademark of Hewlett-Packard Company.

BOSX is a registered trademark of Bull S.A.

IRIX is a registered trademark of Siemens Nixdorf, A.G.

NeXTStep is a registered trademark of the NeXT Computer, Inc.

Purify, Quantify are registered trademarks of Pure Software Inc.

Windows is a registered trademark of Microsoft Corporation.

All other company or product names quoted are trademarks or registered trademarks of their respective trademark holders.

Who should read this manual

This manual describes how to use O_2Search, an O_2 module that enables you to design and develop applications which use a full text index. The module provides indexing and search capabilities for text-based objects in your O_2 database. The manual contains a comprehensive list of O_2Search methods and commands.

Other documents available are outlined, click below.

See O2 Documentation set.

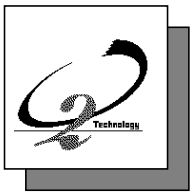


Table of Contents

TABLE OF CONTENTS

1 Introduction 9

1.1 System overview 10

1.2 Document search and retrieval 11

Users' perspective 12
Search engine 12
SEARCH'97 12
O_2Search 13

2 Setting Up O_2Search 15

3 Using O_2Search 17

3.1 Defining document objects in O_2 18

The class Document 18
The method fti_report 19
The O_2 schema 20

3.2 Indexing document objects 21

Creating a document object 21
Opening an O_2Search session 21
Opening an O_2Search collection 22
Indexing a document object 23

3.3 Searching for and retrieving document objects 24

Formulating an O_2Search query 24
Simple Syntax 24
Explicit Syntax 25
Submitting a query to O_2Search 25
Retrieving the result of an O_2Search query 25

4 O_2Search API Reference 27

4.1 O2FTISession Methods 28

O2FTISession 29
~O2FTISession 31
setInfo 32

getInfo 33

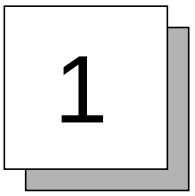
4.2 O2FTICollection Methods 34

O2FTICollection 35
~O2FTICollection 36
reset 37
destroy 38
add 39
update 40
remove 41
optimize 42

4.3 O2FTISearch Methods 43

O2FTISearch 44
~O2FTISearch 45
getFoundDocument 46
getFoundDocumentScore 47
getNumFoundDocuments 48
getNumProcessedDocuments 49

TABLE OF CONTENTS



Introduction

O_2Search combines the power of O_2 with the SEARCH'97 search engine (by Verity, Inc.). This combination enables you to provide full text indexing and search capabilities for any document object in your O_2 database.

This chapter provides an overview of the O_2 system, an introduction to document search and retrieval technology, and a brief overview of O_2Search. See the following sections for more inforamtion:

- Section 1.1, System overview
- Section 1.2, Document search and retrieval

To effectively use this manual, you should have some knowledge of O_2 and O_2's C++ ODMG binding interface. You can learn about SEARCH'97 in Verity's manuals.

1 Introduction

1.1 System overview

O_2's system architecture is illustrated in the following figure:

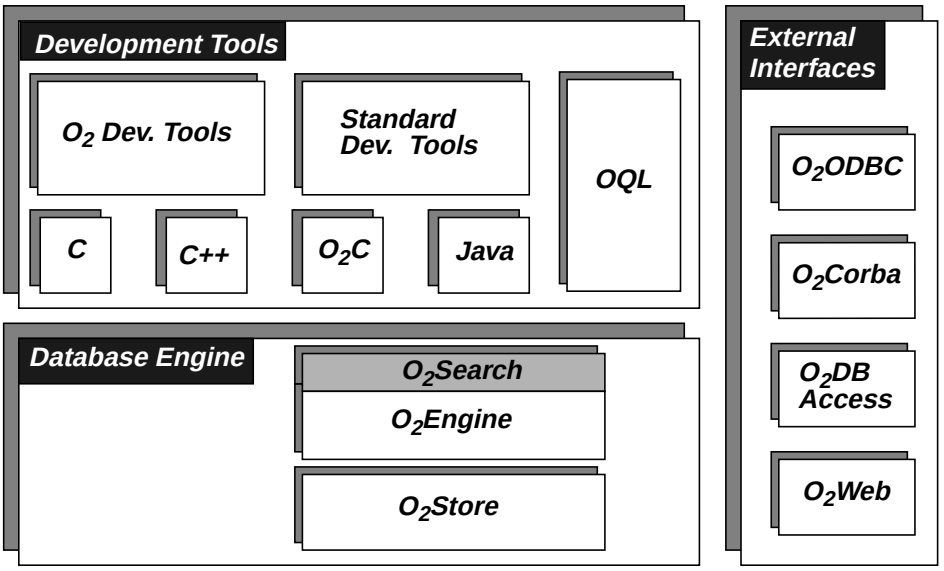


Figure 1.1: O_2 System Architecture

The O_2 system consists of the database engine, development tools, and external interfaces.The database engine provides all the capabilities of a database and an object-oriented system. This engine is accessible by using programming languages, O_2 development tools, and other standard development tools. Numerous external interfaces are also provided.

The database engine consists of the following:

- O_2Store — The database management system (DBMS) provides low-level facilities, through O_2Store API, to access and manage databases: disk volumes, files, records, indexes, and transactions.
- O_2Engine — The object database engine provides direct control of schemas, classes, objects, and transactions through O_2Engine API. It provides full text indexing and search capabilities with O_2Search and spatial indexing and retrieval capabilities with O_2Spatial. It includes a Notification manager for informing other clients connected to the same O_2 server that an event has occurred, a Version manager for handling multiple object versions and a Replication API for synchronizing multiple copies of an O2 system.

Programming Languages:

You can create and manage O_2 objects by using the following programming languages:

- C — O_2 functions can be invoked by C programs.
- C++ — ODMG-compliant C++ binding.
- Java — ODMG-compliant Java binding.
- O_2C — An object-oriented fourth-generation language specifically for developing object database applications.
- OQL — ODMG-standard, SQL-like object query language capable of handling complex O_2 objects and methods.

O_2 Development Tools:

- O_2Graph — Creates, modifies, and edits any type of object graph.
- O_2Look — Designs and develops graphical user interfaces, provides interactive manipulation of complex and multimedia objects.
- O_2Kit — Library of predefined classes and methods for fast development of user applications.
- O_2Tools — Complete graphical programming environment for designing and developing O_2 database applications.

Standard Development Tools:

You can use any standard programming language system, such as Visual C++ and Sun Sparcworks and any supported operating system.

External Interfaces:

- O_2Corba — Creates O_2-Orbix servers for accessing O_2 databases with CORBA.
- O_2DBAccess — Connects O_2 applications to relational databases on remote hosts; invokes SQL statements.
- O_2ODBC — Connects remote ODBC client applications to O_2 databases.
- O_2Web — Creates O_2 World Wide Web servers for accessing O_2 databases through the internet network.

1.2 Document search and retrieval

The proliferation of personal computers and distributed servers, together with the distribution of different forms and types of information across the World Wide Web, has increased the demand for powerful information research and analysis tools.

1 Introduction

Document search engines enable users to search a large number of documents using specific selection criteria. These engines have proliferated across the Web to enhance users' capability to locate and collect information.

The first search engines processed only plain-text documents. Today, they are capable of efficiently processing many different types of documents and supporting user queries that address specific document structures.

Users' perspective

From the user's perspective, the primary interface to a document search engine is the engine's query language, which is used to search documents stored in a document repository. The syntax of document retrieval query languages is usually simple and combines Boolean operators for textual searches (like "**`NEAR`**" or "**`STEM`**") with operators based on the logical structure of documents (like "**`attribute contains`**"). The result of a query is a set of identifiers for documents or regions within documents that satisfy the search criteria. The user can choose a document from the result list to view with an appropriate tool.

Search engine

From the perspective of the search engine, a document repository is a set of indexes. Currently, many different methods of indexing are used. To index a document, the engine analyzes it and builds a set of keywords and keyword locations. The set of keywords and locations comprise the document index. Users' queries are translated into a set of keywords and associated operators that are used to search the document index.

SEARCH'97

SEARCH'97 is a document search and retrieval system developed and distributed by Verity, Inc. Documents maintained by the SEARCH'97 engine are full-text indexed and maintained in a SEARCH'97 collection.

SEARCH'97's knowledge-based indexing and retrieval uses the knowledge representation Verity Query Language to store and organize the information, and to accelerate category-based searching. The search engine also takes advantage of thesauri, word stemming, linguistic analysis, and other methods of query expansion. By using filters and viewers, the engine can extract, index, and render in HTML the contents of documents stored in more than 200 formats. These functions are executed in real time for Web browser display, maintaining the format designed by the document's creator.

The main features of SEARCH'97 include the following:

- a powerful query language that supports word, concept, field, and proximity searches.
- universal data support, including the support to multiple document types and formats located anywhere on the Web.
- dynamic access to documents that are stored in a collection.

An individual SEARCH'97 collection represents a logical group of documents plus a set of meta data about those documents (such as word indexes, document field information, and logical pointers to the actual document text). A collection is an API object that can be searched by the SEARCH'97 engine. Each collection is stored in a file. Concurrent access to document data is enabled through file sharing and is synchronized through file locking.

For more information about SEARCH'97, please refer to Verity's SEARCH'97 documentation.

O_2Search

O_2Search combines the power of O_2 with that of SEARCH'97 to enable application programmers to add full text indexing and search capabilities to any document object stored in an O_2 database. O_2Search supports applications ranging from dedicated noninteractive index builders to responsive GUI front-end interfaces. An application program interface (API) is provided to support the integration of full text index capabilities within O_2 applications.

An O_2 application using O2Search can perform four functions:

- Store documents in objects of the database
- Index document objects to make them searchable
- Search and retrieve document objects that satisfy users' queries
- Maintain and optimize full text indexes

An O_2Search application can be configured to search, index, or maintain indexes. Then, at run time, the application can change the type of operation it is performing or run more than one type of operation at a time. For example, applications can run searches while indexing and performing maintenance functions.

O_2Search applications can use the following search techniques:

- Searching for document objects that contain specific words — Searches can also be based on an expansion list associated with a word such as a list of variations containing the stem word, a list of synonyms, or a list of sound-alike words.

1 Introduction

- Searching for document objects that contain certain words in close proximity, in the same phrase, or in the same paragraph.

O_2Search assigns a score to each retrieved document object, which remains associated with the object for the duration of the search. Retrieved documents are listed in order by score.

An application program interface (API) enables you to integrate full text index capabilities within your O_2 applications.

Three C++ classes make up the O_2Search API: **`O2FTISession`**, **`O2FTICollection`** and **`O2FTISearch`**.

Class **`O2FTISession`** opens a new O_2Search session and records user input.

O_2Search indexes and searches **`O2FTICollection`** instances, which represent full-text indexed SEARCH'97 collections. The contents of documents are stored in O_2, whereas SEARCH'97 collections into which the documents are inserted through the **`O2FTICollection`** API class contain only index and meta data, including a pointer to the appropriate document objects in O_2.

An application can use methods in the **`O2FTISearch`** class to search and retrieve documents stored in full-text indexed collections. These methods produce lists of pointers to documents that satisfy the search criterion. The application uses these pointers to retrieve the corresponding O_2 object in the database. You can further narrow your search by submitting the results to a new search.

All updates to data in O_2 databases are made within transactions, so that updates can be aborted when errors occur. Because SEARCH'97 does not support transactions, O_2Search manages them within the class **`O2FTICollection`**.

Search engines are commonly used in the context of intranet/Internet applications, in which a Web browser is used as the interface for submitting queries and viewing results. The combined use of O_2Search and O_2Web constitutes a powerful and easy to use tool for the development of Web servers on top of O_2.

2 Setting Up O_2Search

After installing O_2 and O_2Search, you can find the following O_2Search-related files in your O_2 home directory. These files are needed to build an O_2Search application:

```
opt/
     O2search/
          /bin
          /common
          /doc
          /filters
          /lib
```

You need to update the **LD_LIBRARY_PATH** environment variable to point to the following directory:

```
$O2HOME/opt/o2search/lib
```

2 Setting Up O2Search

3 Using O_2Search

This chapter presents an example of a simple O_2Search application. The example introduces the main components of the O_2Search architecture and shows how the O_2Search API can be used to build O_2 applications that make use of full-text indexing (FTI) and retrieval capabilities.

See the following sections for more information:

- Section 3.1, Defining document objects in O2
- Section 3.2, Indexing document objects
- Section 3.3, Searching for and retrieving document objects

See Chapter 4, O2Search API Reference, for a complete description of the API. Also, see the sample application in $O2HOME/samples that show how O_2Search can be used in conjunction with O_2Web.

3 Using O_2Search

3.1 Defining document objects in O_2

You must define a C++ *document class* and import it into O_2. Also, you must define a root of persistence in O_2 to store instances of this class; these instances are called *document objects*.

The following example defines a C++ class named **`Document`**. A name **`TheRepository`** of type **`o2_list_Document`** is defined in the schema to be used as the root of persistence for documents. No constraints are imposed on the name or structure of document classes or on the way instances of classes are stored in O_2. The only constraint is that the document class must define the method **`fti_report`**. Your application can define any number of document classes to store and manipulate different types of documents.

The `class` Document

The following code defines the class Document with attribute **`data`** to hold document contents. The other attributes in this class are used to hold additional information about a document and can be ignored for now. You can see how these attributes are used by reviewing the sample

application provided with O_2Search, from which this class defintion is taken.

```
class Document
{
   public:
      d_Bits   data;
      char*    name;
      char*    origin;
      char*    abstract;
      char*    kind;

   public:
      Document () {};
      d_String get_query ();
      d_String get_abstract ();
      d_String html_title ();
      d_Bits html_prolog (char *query, char *userdata);
      d_Bits html_header (char *query, char *userdata);
      d_Bits html_report (char *query, char *userdata);
      d_Bits html_footer (char *query, char *userdata);
      d_Bits fti_report ();
};
```

The method fti_report

Before you can index a class for full-text searching, you must define the **fti_report** method within that class. This method returns the text to be indexed, which may include SGML tags as well as plain text. You can also use this method to customize the text returned from document objects.

The following code defines the method **fti_report** in class **Document.** This code returns the attribute **data**, which holds the contents of the document.

```
d_Bits Document::fti_report ()
{
return this->data;
}
```

The O2 schema

The O_2 schema used in the preceding example is composed of the classes **`Document`** and **`o2_list_Document`**, which are generated by importing the C++ class **`Document`** and the type list **`Document`**. In addition, the following name is defined:

```
constant name TheRepository:o2_list_Document;
```

This name is the only root of persistence defined in the schema for this example. It is used to store instances of class **`Document`**.

3.2 Indexing document objects

This section shows how to create a document object, store it in O_2, and use the O_2Search API to index it. The complete code follows:

```
d_Ref<Document> doc = new Document;
doc->name = “my_document”;
d_Bits doc_data = str;
doc->data = doc_data;

transaction.begin();
try {
    // Creates a session
    O2FTISession  FTIsession (argc,argv);

    // Creates a full text collection.
    O2FTICollection  collection
    (FTIsession,“collections/example”);

    // Add the new document in the full text index.
    collection.add (doc.o2_get_handle ());

    // Add the new document in O2.
    d_List <d_Ref <Document> >  repository(“TheRepository”);
    repository.insert_element_last(doc);

    transaction.validate();

} catch (const O2FTIException &e) {
     cout << "Error while indexing document: " << e.errorCode()
```

Creating a document object

Before executing the preceding example, you must create a new document and store it in the O_2 using the C++ ODMG interface. The example assumes that you have stored the document contents in the variable **`str`**, of type **`char *`**. This may be, for example, the contents of a Microsoft Word or Acrobat PDF file.

Opening an O2Search session

To be able to index a document object, an O_2Search session must be open. The constructor of the class **`O2FTISession`** is used to open a new session. In the example in this chapter, the default settings of a

session are used, but you can change them during the session by using the public interface of class **O2FTISession**. For example, you can turn on or off logging of errors related to indexing and maintenance at run time.

O2FTISession records user input, effectively emulating a session. Most applications require you to open at least one session, but you can open multiple sessions at the same time. For example, if an application supports multiple users, a separate session can be created for each user.

Opening an O2Search collection

An instance of **O2FTICollection** is a transient object that corresponds to a SEARCH'97 collection. The constructor of the class **O2FTICollection** either opens the existing collection or creates a new one if none of the given name is found. The constructor name parameter is used to locate the corresponding SEARCH'97 collection file or to create a new one. You must provide the absolute path, including the name.

The **O2FTICollection** class contains the internal data structure required to enable updates to SEARCH'97 to be managed in the context of an O_2Search transaction. When an O_2Search application updates an instance of **O2FTICollection**, the update of the corresponding SEARCH'97 collection is performed only after transaction validation is complete. In the same way, any document removed from a full-text indexed collection through the **O2FTICollection** interface belongs to that collection until the transaction is validated. The commit or validate of an O_2 transaction in an O_2Search application synchronizes the contents of the document objects stored in the database with those of the SEARCH'97 collections used in a given session.

Multiple applications can work on the same O_2Search collection and documents at the same time. The O_2 transaction locking mechanism enables multiple applications to generate indexes for documents in a collection while searching the same collection without the risk of losing any indexing information.

Multiple **O2FTICollection** objects can be used at the same time. Maintenance methods are also defined within **O2FTICollection** objects for optimization and garbage collection.

Indexing a document object

To index a document object, add the document to an instance of **`O2FTICollection`**. When an O_2 transaction is committed or validated, the contents of the document object (given by method **`fti_report`**) are indexed and the corresponding collection and document in the SEARCH'97 collection are updated.

3.3 Searching for and retrieving document objects

The following code demonstrates using the class **O2FTISearch** to query indexed collections.

```
char  *query = “O2 <NEAR> VERITY OR O2SEARCH”;
try {
   // Open a full text session.
   O2FTISession  FTIsession (argc,argv);

   // Open or Create a full text collection.
   O2FTICollection  collection
(FTIsession,“collections/example”);

   // Open a Search object
   O2FTISearch  search (FTIsession, collection, query);

   cout << search.getNumFoundDocuments () <<  " of "
        << search.getNumProcessedDocuments ()
        << " documents matched query (" << query << ")\n";
   cout << "\n";
   for (int i=0; i < search.getNumFoundDocuments (); i++) {
        cout << "\n";
        cout << “Score ” << search.getFoundDocumentScore(i);
        d_Ref<Document> doc(search.getFoundDocument(i),
OL_READ);
        cout << doc->name << "\n";
   }
```

Formulating an O_2Search query

A query expression is any statement you enter for performing a search. Two kinds of full-text queries are defined: simple syntax and explicit syntax.

- ***Simple Syntax***

When you use simple syntax, the query parser interprets single words as if they were preceded by the MANY modifier and the STEM operator. When you use the MANY keyword, the score for each document is

based on the frequency with which the word appears in the document; the more times it appears, the higher the document's score.

The full text engine searches for the words you specify, as well as words that have the same stem. The query **"film"** retrieves all documents containing the word film and all words with the same stem. For example, films, filmed, and filming are stemmed variations of the word film. The query **"cartoons Jones Chuck"** retrieves documents about cartoons made by Chuck Jones. The implicit operator **AND** is added between words.

When you enclose a word in double quotations marks, the word is interpreted literally, and no stemmed variations are included in the search.

In this example, a full-text query is formulated using the explicit syntax and the operator **near**.

- ***Explicit Syntax***

Explicit syntax uses the SEARCH'97 query language. A full description of its syntax and usage is provided in the Verity Search'97 Developers Kit API Reference Guide.

An example of an explicit syntax query follows:

```
<paragraph> ("software", <sentence> ("merge",
<phrase> ("annual", "purchase")))
```

For more information on the query language, please refer to the SEARCH'97 documentation.

Submitting a query to O2Search

When you create an **O2FTISearch** object, the query is submitted to O_2Search. The constructor expects as arguments the session, the collection on which the query is to be executed, and the text of the query. The O_2Search engine executes the query, retrieving all documents in the collection that satisfy the query criteria. In the preceding example, the selection criteria is based on the proximity of the word O2 to the words VERITY or O2SEARCH.

Retrieving the result of an O2Search query

Several methods are defined in class **O2FTISearch**, some of which are used in the preceding example. The method **getFoundDocument**, for

example, enables applications to retrieve a document by specifying the ordinal number of the document in a list of retrieved documents.

You can limit the results of a search to one of the following:

- documents whose relevance score is above a given threshold,
- the top scoring *n*th documents.

You can also submit search results as a new search object to further narrow the search.

4 O_2Search API Reference

The O_2Search API enables users to use O_2Search indexing and searching facilities for any type of document.

When an error is detected, O_2Search throws a C++ exception containing an error code and description.

This chapter describes the public interface of the C++ classes that make up the O2Search API. The following classes are included:

- Section 4.1, O2FTISession Methods
- Section 4.2, O2FTICollection Methods
- Section 4.3, O2FTISearch Methods

4 O2Search API Reference

4.1 O2FTISession Methods

The public interface of the class is defined as follows:

```
class O2FTISession
   {
      public:
         typedef enum { None    = 0,
             Fatal   = 0x01,
             Error   = 0x02,
             Warn    = 0x04,
             Status  = 0x08,
             Info    = 0x10,
             Verbose = 0x20,
             Debug   = 0x40,
             Basic   = Fatal | Error | Warn | Status,
             All     = Basic | Info | Verbose
    } MessageLevel;

    O2FTISession (int      argc,
             const char**  argv,
             const char*   locale   = “english”,
             MessageLevel  outLevel = Fatal,
             MessageLevel  logLevel = Fatal,
             const char*   logFile  = NULL)
             throw (O2FTIException);
    ~O2FTISession () throw (O2FTIException);

    void setInfo (MessageLevel  outLevel,
             MessageLevel  logLevel,
             const char*   logFile) throw (O2FTIException);

    void getInfo (MessageLevel* outLevel,
             MessageLevel* logLevel,
             char**        logFile) throw (O2FTIException);
   };
```

O2FTISession

Description — Constructor. Creates an O_2Search session and returns an **O2FTISession** instance. Multiple sessions can be created by an application.

Syntax

```
O2FTISession::O2FTISession (
int                          argc,
char**                       argv,
char*                        locale   = "english",
O2FTISession::MessageType    outLevel = Fatal,
O2FTISession::MessageType    logLevel = Fatal,
char*                        logFile  = NULL)
```

Parameters

argc	Number of elements in **argv**.
argv	Pointer to the array of **main** function arguments
locale	Language driver to be used instead of the standard American English language.
outLevel	Types of messages generated by the FTI engine. Types are listed under Comments.
loglevel	Error message types stored in the log file. Types are listed under Comments.
logFile	Absolute path for the log file, including file name.

Returns

When successful, returns a new **O2FTISession** object.

When unsuccessful, throws an **O2FTIException**.

Comments

Message types that may be supplied in the parameter **outLevel** include the following. Error messages are stored in the log file.

- Fatal — Fatal error
- Error — Common error
- Warn — Warnings
- Status — Engine status information
- Info — Detailed information
- Verbose — Verbose information
- Debug — Debugging information
- Basic — Basic information; a combination of fatal and common errors, warnings and status information

- All — All information; includes all preceding message categories except debugging

~O2FTISession

Description Destructor. Terminates the session. Frees all resources associated with the session including search objects and allocated memory.

Syntax `O2FTISession::~O2FTISession ()`

Parameters None

Returns Always frees the session. No error is returned.

setInfo

Description Modifies the state of a session.

Syntax

```
void O2FTISession::setInfo (
O2FTISession::MessageType outLevel = Fatal,
O2FTISession::MessageType logLevel = Fatal,
char*                     logFile  = NULL)
```

Parameters

`outLevel` Types of messages generated by the FTI engine. Types are listed under Comments.

`loglevel` Error message types stored in the log file. Types are listed under Comments

`logFile` Absolute path for the log file, including file name

Returns When successful, returns nothing.

When unsuccessful, throws an **`O2FTIException`**.

Comments Message types that may be supplied in the parameter **`outLevel`** include the following. Error messages are stored in the log file.

- Fatal — Fatal error
- Error — Common error
- Warn — Warnings
- Status — Engine status information
- Info — Detailed information
- Verbose — Verbose information
- Debug — Debugging information
- Basic — Basic information; a combination of fatal and common errors, warnings and status information
- All — All information; includes all preceding message categories except debugging

getInfo

Description Returns information about the current session.

Syntax

```
void O2FTISession::getInfo (
O2FTISession::MessageType  *outLevel,
O2FTISession::MessageType  *logLevel,
char**                      logFile)
```

Parameters

`outLevel` Types of messages generated by the FTI engine. Types are listed under Comments.

`loglevel` Error message types stored in the log file. Types are listed under Comments

`logFile` Absolute path for the log file, including file name

Returned pointers refer to internal data, which you must not modify. You must make a copy if you need to modify them.

Returns When successful, returns nothing.

When unsuccessful, returns an **`O2FTIException`**.

Comments Error message values that may be supplied in the parameter **`outLevel`** include the following. Error messages are stored in the log file.

- Fatal — Fatal error
- Error — Common error
- Warn — Warnings
- Status — Engine status information
- Info — Detailed information
- Verbose — Verbose information
- Debug — Debugging information
- Basic — Basic information; a combination of fatal and common errors, warnings and status information
- All — All information; includes all preceding message categories except debugging

4.2 O2FTICollection Methods

The public interface of the class is defined as follows:

```
class O2FTICollection
{
  public:
    O2FTICollection (const O2FTISession&  session,
                     const char*    name) throw       (O2FTIException);
    O2FTICollection (const O2FTICollection&  copy);
    ~O2FTICollection () throw (O2FTIException);
    const O2FTICollection& operator= (const O2FTICollection&
                         collection);
    static O2FTICollection* getCollection (const char*   collectionName);
    const O2FTISession& getSession () const;
    void reset () throw (O2FTIException);
    void optimize () throw (O2FTIException)
    void destroy ();

    void add     (int          numDocuments,
                  const Handle*  hdDocumentArray) throw  (O2FTIException);
    void remove (int           numDocuments,
                  const Handle*  hdDocumentArray) throw  (O2FTIException);
    void update (int           numDocuments,
                  const Handle*  hdDocumentArray) throw  (O2FTIException);
    void add    (const Handle   hdDocument) throw       (O2FTIException);
    void remove (const Handle   hdDocument) throw       (O2FTIException);
    void update (const Handle   hdDocument) throw       (O2FTIException);
  };
```

O2FTICollection

Description Constructor. Opens a SEARCH'97 collection of full-text indexed document objects, or creates a new empty collection of the same name if none exists.

Syntax

```
O2FTICollection::O2FTICollection (
const O2FTISession*  session
char*                name)
```

Parameters

session Current session.

name Name of the opened collection. If an absolute path name is not provided, the current directory is used.

Returns When successful, returns a new **O2FTICollection** object.

When unsuccessful, returns an **O2FTIException** object containing the error **Unable to create a new collection**, indicating that disk space was insufficient to open the collection.

~O2FTICollection

Description Destructor. Closes the open collection.

Syntax `O2FTICollection::~O2FTICollection ()`

Parameters None.

Returns Always closes the SEARCH'97 collection and deletes the corresponding C++ object. No error is returned.

reset

Description Deletes all document objects from the collection and empties and deletes the indexes.

Syntax `void O2FTICollection::reset ()`

Parameters None.

Returns When successful, empties the collection and deletes the indexes.

When unsuccessful, throws an **`O2FTIException`**.

destroy

Description Resets and closes the collection, then deletes it. The deleted collection is no longer available for searching or indexing.

Syntax `void O2FTICollection::destroy ()`

Parameters None.

Returns Always frees the **`O2FTICollection`**. Returns no errors.

add

Description Inserts the specified list of document objects into a SEARCH'97 collection, or creates a new object of the given name if none exists. O_2Search inserts the new document object into the SEARCH'97 collection and updates the corresponding full-text indexes only when the O_2 transaction is committed or validated.

Syntax

```
void O2FTICollection::add (int numDocuments,
                           const Handle*  documentArray)
void O2FTICollection::add (const Handle   document)
```

Parameters

numDocuments	Number of document objects in **documentArray**.
documentArray	Document object handle list.
document	Document object handle.

Returns When successful, inserts and indexes the listed document objects.

When unsuccessful, throws an **O2FTIException** object containing the message **error during modification of the index.**

update

Description Updates the specified list of document objects in a collection. O_2Search updates the document objects in the SEARCH'97 collection and updates the corresponding full-text indexes only when the O_2 transaction is committed or validated.

If an update is requested for a document that does not already exist in the collection, O_2 creates it.

Syntax

```
void O2FTICollection::update (int
numDocuments,const Handle*  documentArray)
void O2FTICollection::update (const Handle*
document)
```

Parameters

numDocuments	Number of document objects in **documentArray**.
documentArray	Document handle list.
document	Document handle.

Returns When successful, updates the listed of document objects and the corresponding index.

When unsuccessful, throws an **O2FTIException** object containing the message **error during modification of the index**.

remove

Description Deletes the specified list of document objects from a document collection. O_2Search removes the document objects from the SEARCH'97 collection and updates the corresponding full-text indexes only when the O_2 transaction is committed or validated.

Syntax

```
void O2FTICollection::remove (int
numDocuments, const Handle*  documentArray)
void O2FTICollection::remove (const Handle*
document)
```

Parameters

numDocuments	Number of document object handles in **documentArray**.
documentArray	Document object handle list.
document	Document object handle.

Returns When successful, deletes the listed document objects and removes all associated index information.

When unsuccessful, throws an **O2FTIException** object containing the message **error during modification of the index**.

optimize

Description Optimizes the structure and contents of the full-text index associated with the collection. You should call this method after all indexing has been completed, because index optimization speeds searches and reduces disk space usage.

Syntax `void O2FTICollection::optimize ()`

Parameters None.

Returns When successful, optimizes the collection.

When unsuccessful, throws an **`O2FTIException`**.

4.3 O2FTISearch Methods

The public interface of the class is defined as follows:

```
class O2FTISearch
   {
public:
   O2FTISearch (const O2FTISession&session,
   const  O2FTICollection&     collection,
   const  char*                query,
   in     threshold  = 0, // between 0 and 100.
   int    numMaxDocs = 0) throw (O2FTIException); // 0 = no limit.

O2FTISearch (const O2FTISession&session,
    const unsigned             numCollections,
    const O2FTICollection*     collections,
    int   threshold  = 0, // between 0 and 100.
    int   numMaxDocs = 0) throw (O2FTIException) // 0 = no limit.

O2FTISearch (const O2FTISession&session,
    const unsigned             numDocuments,
    const Handle*              documents,
    const char*                query,
    int   threshold  = 0,      // between 0 and 100.
    int   numMaxDocs = 0) throw (O2FTIException); // 0 = no limit.

O2FTISearch (const O2FTISession& session, // Narrow query result
    const O2FTISearch&         search,
    const char*                query)
                               throw (O2FTIException);

O2FTISearch (const O2FTISearch&  search);
~O2FTISearch ();

Handle getFoundDocument  (int nth) throw  (O2FTIException);
int getFoundDocumentScore (int nth) throw  (O2FTIException);

int getNumFoundDocuments ();
int getNumProcessedDocuments ();

};
```

4 O2Search API Reference

O2FTISearch

Description Constructor. Searches a collection for document objects that meet the specified search criteria.

You can submit a query to one or more collections, and you can narrow the search by submitting an existing **O2FTISearch** object to **O2FTISearch**.

Syntax

```
O2FTISearch::O2FTISearch (
          const O2FTISession&    session,
          const O2FTICollection& collection,
          const char*            query,
          const int              threshold  = 0,
          const int              numMaxDocs = 0)
O2FTISearch::O2FTISearch (
          const O2FTISession&    session,
          const unsigned         numCollections,
          const O2FTICollection* collectionArray,
          const char*            query,
          const int              threshold  = 0,
          const int              numMaxDocs = 0)
O2FTISearch::O2FTISearch (
          const O2FTISession&    session,
          const O2FTISearch      oldSearch,
          const char*            newQuery)
```

Parameters

session	Session in which the search is to be performed.
collection	Searched collection.
collectionArray	List of searched collections.
numCollections	Number of collections in **collectionArray**.
query, **newQuery**	The query in Verity syntax.
oldSearch	**O2FTISearch** object for which the search is to be narrowed.
threshold	Minimum score required for a documents to be retrieved.
numMaxDocs	Maximum number of documents to be returned.

Returns When successful, creates a new **O2FTISearch** object containing the documents satisfying the search requirements, sorted by relevance score.

When unsuccessful, throws an **O2FTIException** object containing the message **incorrect query.**

~O2FTISearch

Description Destructor. Removes the **`O2FTISearch`** object.

Syntax **`O2FTISearch::~O2FTISearch()`**

Parameters None

Returns Nothing

getFoundDocument

Description Retrieves the handle for the *n*th (zero-based) document object returned in the document list of the **`O2FTISearch`** object.

Syntax **`Handle O2FTISearch::getFoundDocument (int  nth)`**

Parameters **`nth`** Ordinal number for the document in the list of retrieved documents.

Returns When successful, returns the requested document object

When unsuccessful, throws an **`O2FTIException`** object containing the message **`index out of range`**.

getFoundDocumentScore

Description Returns the relevance score of the *n*th document object in the document list of the **`O2FTISearch`** object.

Syntax **`int O2FTSearch::getFoundDocumentScore (int nth)`**

Parameters **`nth`** Ordinal number for the document in the list of retrieved documents.

Returns When successful, returns the corresponding document's score.

When unsuccessful, throws an **`O2FTIException`** object containing the message **`index out of range`**.

getNumFoundDocuments

Description Returns the number of document objects that satisfy the search criteria.

Syntax `int O2FTISearch::getNumFoundDocuments ()`

Parameters None.

Returns When successful, returns the number of document objects that meet the search criteria.

When unsuccessful, throws an **`O2FTIException`**.

getNumProcessedDocuments

Description Returns the number of document objects processed during the search.

Syntax `int O2FTISearch::getNumProcessedDocuments ()`

Parameters None.

Returns When successful, returns the number of document objects processed.

When unsuccessful, throws an **`O2FTIException`**.

4 O2Search API Reference